Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Computación Gráfica e Interacción Humano Computadora

GRUPO: 03

Semestre 2022-1

Manual de Técnico

Fecha de Entrega: 09/12/21

Integrante de equipo:

1. Meza Vega Hugo Adrian

Número de cuenta: 314344580

Email: hamv1509@gmail.com

Grupo de teoría: 03

Parte 1: Elementos por incluir dentro del escenario.

1. **Geometría**

Para la geometría del escenario utilicé distintos modelos con sus correspondientes texturas.

En su mayoría, los modelos los recuperé de la página: www.RigModels.com y son de uso gratuito.

Cuentan con licencia para uso exclusivo de estudiantes y cuestiones académicas.

A continuación, muestro la carga de modelos:

```
562     pasto = Model();
563     pasto.LoadModel("Models/pasto.obj");
564     arboles = Model();
565     arboles.LoadModel("Models/arbol.obj");
566     arboles2 = Model();
567     arboles2.LoadModel("Models/arbol2.obj");
568     arbolSeco = Model();
569     arbolSeco.LoadModel("Models/arbolSeco.obj");
570     roca = Model();
571     roca.LoadModel("Models/roca.obj");
572
573     tronco = Model();
574     tronco.LoadModel("Models/tronco.obj");
575
576     carreta = Model();
577     carreta.LoadModel("Models/carreta.obj");
578     carreta2 = Model();
579     carreta2.LoadModel("Models/carreta2.obj");
580
581     dragon = Model();
582     dragon.LoadModel("Models/dragon.obj");
583
584     ave1 = Model();
585     ave1.LoadModel("Models/ave.obj");
586     ave2 = Model();
587     ave2.LoadModel("Models/ave.obj");
588
589
590     casaShrek = Model();
591     casaShrek.LoadModel("Models/casaShrekSinPuerta.obj");
592     puertaCasaShrek = Model();
593     puertaCasaShrek.LoadModel("Models/casaShrekPuerta.obj");
594
595     shrekCuerpo = Model();
596     shrekCuerpo.LoadModel("Models/ShrekB.obj");
597     shrekAL = Model();
598     shrekAL.LoadModel("Models/ShrekAL.obj");
599     shrekAR = Model();
600     shrekAR.LoadModel("Models/ShrekAR.obj");
601     shrekLL = Model();
602     shrekLL.LoadModel("Models/ShrekLL.obj");
603     shrekLR = Model();
604     shrekLR.LoadModel("Models/ShrekLR.obj");
605
```

a) Muñeco de nieve.

Este modelo lo realicé a partir de figuras primitivas vistas previamente en las clases de laboratorio.

Este modelo consiste en 3 esferas de diferentes radios, de la mas grande a la más pequeña, además de dos esferas para los ojos, una pirámide para la nariz y otras dos más para los brazos. Se hace uso de modelado jerárquico.

```
2343        //cuerpo
2344        model = glm::mat4(1.0);
2345        model = glm::translate(model, glm::vec3(20.0f, 0.0f, -10.0f));
2346        modelaux = model;
2347        model = glm::scale(model, glm::vec3(2.50f, 2.5f, 2.5f));
2348        model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
2349        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
2350        Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
2351        nieveTexture.UseTexture();
2352        muneco.render();
2353        model = modelaux;
2354        model = glm::translate(model, glm::vec3(0.0f, 3.0f, 0.0f));
2355        modelaux2 = model;
2356        modelaux = model;
2357        model = glm::scale(model, glm::vec3(1.8f, 1.8f, 1.8f));
2358        model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
2359        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
2360        Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
2361        nieveTexture.UseTexture();
2362        muneco.render();
2363        //brazos
2364        model = modelaux2;
2365        model = glm::translate(model, glm::vec3(1.6f, 0.2f, -0.9f));
2366        model = glm::scale(model, glm::vec3(3.75f, 0.25f, 0.25f));
2367        model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
2368        model = glm::rotate(model, -90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
2369        model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
2370        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
2371        Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
2372        maderaTexture.UseTexture();
2373        meshList[1]->RenderMesh();
2374
2375        model = modelaux2;
2376        model = glm::translate(model, glm::vec3(1.6f, 0.2f, 0.9f));
2377        model = glm::scale(model, glm::vec3(3.75f, 0.25f, 0.25f));
2378        model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
2379        model = glm::rotate(model, -90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
2380        model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
2381        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
2382        Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
2383        maderaTexture.UseTexture();
2384        meshList[1]->RenderMesh();
2385
2386        model = modelaux;
2387        model = glm::translate(model, glm::vec3(0.0f, 2.5f, 0.0f));
2388        modelaux = model;
2389        model = glm::scale(model, glm::vec3(1.25f, 1.25f, 1.25f));
2390        model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
2391        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
2392        Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
2393        nieveTexture.UseTexture();
2394        muneco.render();
```

```
1395        //ojos
1396        model = modelaux;
1397        model = glm::translate(model, glm::vec3(1.0f, 0.6f, -0.5f));
1398        model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
1399        model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
1400        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
1401        Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1402        negroTexture.UseTexture();
1403        muneco.render();
1404        model = modelaux;
1405        model = glm::translate(model, glm::vec3(1.0f, 0.6f, 0.5f));
1406        model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
1407        model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
1408        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
1409        Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1410        negroTexture.UseTexture();
1411        muneco.render();
1412        //Nariz
1413        model = modelaux;
1414        model = glm::translate(model, glm::vec3(1.6f, 0.2f, 0.0f));
1415        model = glm::scale(model, glm::vec3(0.75f, 0.25f, 0.25f));
1416        model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
1417        model = glm::rotate(model, -90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
1418        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
1419        Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
1420        naranjaTexture.UseTexture();
1421        meshList[1]->RenderMesh();
1422
```

b) Casa de shrek

```
//Casa
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(90.0f, -2.25f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
casaShrek.RenderModel();
```

c) Arboles

Organizados y editados para implementarlos directamente en el proyecto.

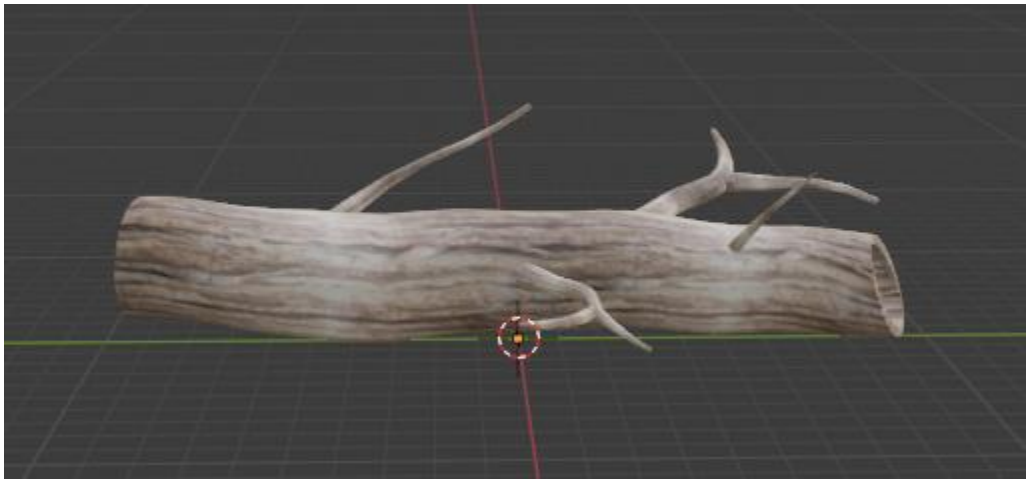
Se hace un arreglo de árboles en todo el perímetro del plano.

```
1736        //arboles
1737            //Parte forntal x-
1738    model = glm::mat4(1.0);
1739    model = glm::translate(model, glm::vec3(-150.0f, 0.0f, 0.0f));
1740    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
1741    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
1742    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1743    arboles2.RenderModel();
1744    model = glm::mat4(1.0);
1745    model = glm::translate(model, glm::vec3(-150.0f, 0.0f, -110.0f));
1746    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
1747    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
1748    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1749    arboles.RenderModel();
1750    model = glm::mat4(1.0);
1751    model = glm::translate(model, glm::vec3(-150.0f, 0.0f, 110.0f));
1752    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
1753    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
1754    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1755    arboles.RenderModel();
1756            //Parte lateral z-
1757    model = glm::mat4(1.0);
1758    model = glm::translate(model, glm::vec3(-105.0f, 0.0f, -155.0f));
1759    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
1760    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
1761    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
1762    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1763    arboles.RenderModel();
1764    model = glm::mat4(1.0);
1765    model = glm::translate(model, glm::vec3(-15.0f, 0.0f, -155.0f));
1766    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
1767    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
1768    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
1769    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1770    arboles2.RenderModel();
1771    model = glm::mat4(1.0);
1772    model = glm::translate(model, glm::vec3(0.30f, 0.0f, -155.0f));
1773    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
1774    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
1775    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
1776    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1777    arboles2.RenderModel();
1778    model = glm::mat4(1.0);
1779    model = glm::translate(model, glm::vec3(100.0f, 0.0f, -155.0f));
1780    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
1781    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
1782    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
1783    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
1784    arboles.RenderModel();
1785
1786            //parte lateral z+
1787    model = glm::mat4(1.0);
1788    model = glm::translate(model, glm::vec3(-105.0f, 0.0f, 155.0f));
```

Otro tipo de árbol es el siguiente:

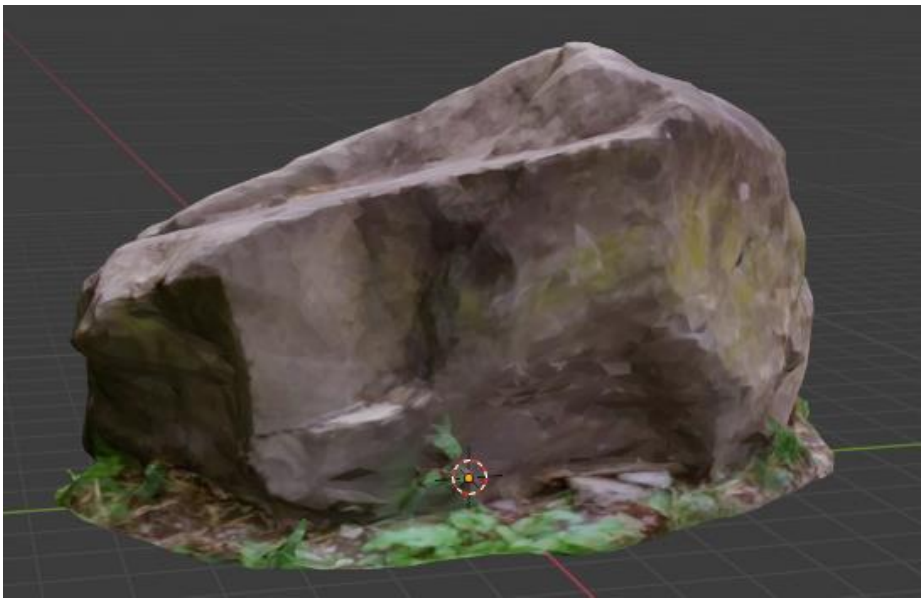
```
        //Arbol seco
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(60.0f, 0.0f, -30.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
arbolSeco.RenderModel();

        //tronco
```

d) Tronco seco:



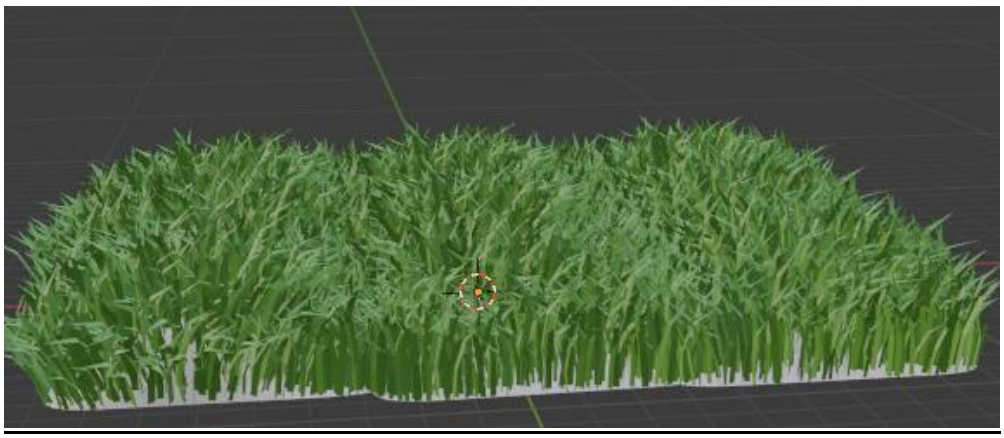
```
        //tronco
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(45.0f, -2.0f, 35.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
tronco.RenderModel();
```

e) Roca



```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-15.0f, -2.0f, 100.0f));
model = glm::scale(model, glm::vec3(3.5f, 3.5f, 3.5f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
roca.RenderModel();
```

f) <u>Pasto:</u>



Se realizaron dos tapetes de pasto en ambos lados de la casa de shrek.

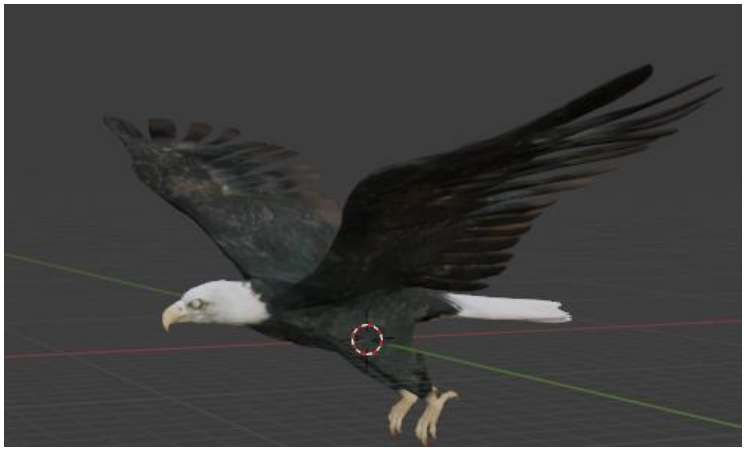
```
        //Pasto
    //lado derecho de la casa
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(75.0f, -1.80f, -25.0f));
    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
    pasto.RenderModel();
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(70.0f, -1.80f, -25.0f));
    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
    model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
    pasto.RenderModel();
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(65.0f, -1.80f, -25.0f));
    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
    pasto.RenderModel();
```

g) Dragon:

```
//************      Dragon      ************

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(desplazamientoDragon));
model = glm::scale(model, glm::vec3(0.40f, 0.40f, 0.40f));

if (avanzad) {
    model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
else {
    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
dragon.RenderModel();
```

Por medio de blender, se realizó la separación de las alas para posteriormente implementar jerarquía y animación de alas al volar.

h) Águilas:



```
//****      Aves        *****
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(desplazamientoAve1));
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
if (avanzaA) {
    model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
else {
    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
ave1.RenderModel();
```

i) Carretas con personajes:

La implementación de estos elementos se analizará en la parte de animación, ya que su código es bastante extenso.

j) Personaje Shrek:

Este modelo se implementó con la ayuda del modelado jerárquico, ya que fue necesario separar el modelo en 5 partes, brazos, piernas y dorso.

```
//**********************       Personaje principal     **********************
model = glm::mat4(1.0);
shrekInicio = glm::vec3(110.0f, 1.0f, 0.0f);
desplazamientoShrek = glm::vec3(shrekInicio.x+ mainWindow.getmuevex(), shrekInicio.y, shrekInicio.z);
if (desplazamientoShrek.x>0) {
    model = glm::translate(model, glm::vec3(desplazamientoShrek));
}
else {
    model = glm::translate(model, glm::vec3(0.0f, 1.0f, 0.0f));
}
modelaux = model;
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
shrekCuerpo.RenderModel();
```

```
//Brazo derecho
model = modelaux;
model = glm::translate(model, glm::vec3(0.045f, 1.35f, -0.9f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
if (mainWindow.getAnimated() == true) {
    if (rotar1 <= 30 and ida1 == true) {
        rotar1 = rotar1 + 2.0;
        if (rotar1 == 30) {
            ida1 = false;
        }
    }
    else {
        if (rotar1 >= -30) {
            rotar1 = rotar1 - 2.0;
            if (rotar1 == -30) {
                ida1 = true;
            }
        }
    }
    model = glm::rotate(model, rotar1 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
}
else {
    model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
shrekAR.RenderModel();
```

```
//Brazo Izquierdo
model = modelaux;
model = glm::translate(model, glm::vec3(0.025f, 1.33f, 0.8f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
if (mainWindow.getAnimated() == true) {
    if (rotar2 <= 30 and ida2 == true) {
        rotar2 = rotar2 + 2.0;
        if (rotar2 == 30) {
            ida2 = false;
        }
    }
    else {
        if (rotar2 >= -30) {
            rotar2 = rotar2 - 2.0;
            if (rotar2 == -30) {
                ida2 = true;
            }
        }
    }
    model = glm::rotate(model, rotar2 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
}
else {
    model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
}
//model = glm::rotate(model, -35 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
shrekAL.RenderModel();
```

2. Avatar.

El avatar es el personaje de shrek, el cual ya mencioné que fue necesario separarlo en 5 partes para poder animarlo de una forma natural.

En la pieza de código vemos como cada una de las extremidades hereda del tronco y de su posición. Éste se mueve al presionar la tecla Y, y la posición de los brazos y piernas, así como su rotación, se ejecuta al mover al personaje.

```
//**********************       Personaje principal     **********************
model = glm::mat4(1.0);
shrekInicio = glm::vec3(110.0f, 1.0f, 0.0f);
desplazamientoShrek = glm::vec3(shrekInicio.x+ mainWindow.getmuevex(), shrekInicio.y, shrekInicio.z);
if (desplazamientoShrek.x>0) {
    model = glm::translate(model, glm::vec3(desplazamientoShrek));
}
else {
    model = glm::translate(model, glm::vec3(0.0f, 1.0f, 0.0f));
}
modelaux = model;
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
shrekCuerpo.RenderModel();
```

```
//Brazo derecho
model = modelaux;
model = glm::translate(model, glm::vec3(0.045f, 1.35f, -0.9f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
if (mainWindow.getAnimated() == true) {
    if (rotar1 <= 30 and ida1 == true) {
        rotar1 = rotar1 + 2.0;
        if (rotar1 == 30) {
            ida1 = false;
        }
    }
    else {
        if (rotar1 >= -30) {
            rotar1 = rotar1 - 2.0;
            if (rotar1 == -30) {
                ida1 = true;
            }
        }
    }
    model = glm::rotate(model, rotar1 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
}
else {
    model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
shrekAR.RenderModel();
```

```
        //Brazo Izquierdo
    model = modelaux;
    model = glm::translate(model, glm::vec3(0.025f, 1.33f, 0.8f));
    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
    if (mainWindow.getAnimated() == true) {
        if (rotar2 <= 30 and ida2 == true) {
            rotar2 = rotar2 + 2.0;
            if (rotar2 == 30) {
                ida2 = false;
            }
        }
        else {
            if (rotar2 >= -30) {
                rotar2 = rotar2 - 2.0;
                if (rotar2 == -30) {
                    ida2 = true;
                }
            }
        }
        model = glm::rotate(model, rotar2 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    }
    else {
        model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    }
    //model = glm::rotate(model, -35 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
    shrekAL.RenderModel();
        //Pierna Izquierda
```

```
        //Pierna Izquierda
    model = modelaux;
    model = glm::translate(model, glm::vec3(0.0f, -0.85f, 0.48f));
    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
    if (mainWindow.getAnimated() == true) {
        if (rotar1 <= 30 and ida1 == true) {
            rotar1 = rotar1 + 2.0;
            if (rotar1 == 45) {
                ida1 = false;
            }
        }
        else {
            if (rotar1 >= -30) {
                rotar1 = rotar1 - 2.0;
                if (rotar1 == -30) {
                    ida1 = true;
                }
            }
        }
        model = glm::rotate(model, rotar1 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    }
    else {
        model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    }
    //model = glm::rotate(model, -35 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
    shrekLL.RenderModel();
        //Pierna derecha
    model = modelaux;
    model = glm::translate(model, glm::vec3(0.0f, -0.85f, -0.56f));
    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
    if (mainWindow.getAnimated() == true) {
        if (rotar2 <= 30 and ida2 == true) {
            rotar2 = rotar2 + 2.0;
            if (rotar2 == 30) {
                ida2 = false;
            }
        }
        else {
            if (rotar2 >= -30) {
                rotar2 = rotar2 - 2.0;
                if (rotar2 == -30) {
                    ida2 = true;
                }
            }
        }
        model = glm::rotate(model, rotar2 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    }
    else {
        model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    }
    //model = glm::rotate(model, -35 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
```

3. Recorrido.

Para el recorrido se utilizaron dos cámaras, una que está ligada al plano del piso y no puede subir mas allá de una determinada altura y otra cámara que es libre. El cambio entre cámaras se hace con la tecla “T”.

```
//Recibir eventos del usuario
glfwPollEvents();
camera.keyControl(mainWindow.getsKeys(), deltaTime, 10.0f ,mainWindow.getTerceraPersona());
camera.mouseControl(mainWindow.getXChange(), mainWindow.getYChange());
```

Hay una variable booleana en window.h que se llama terceraPersona y se activa o desactiva con la tecla “T”. El método mainWindow.getTercesarPersona() devuelve el valor de esta variable y es lo que permite hacer el cambio entre cámaras.

```
//Cambio de camara
if (key == GLFW_KEY_T and action == GLFW_PRESS)
{
    if (theWindow->terceraPersona == true) {
        theWindow->terceraPersona = false;
    }
    else {
        theWindow->terceraPersona = true;
    }
}
```

En el archivo de Camera.cpp se hizo una especificación para que dependiendo del valor de la variable de posición, hay un conjunto de coordenadas posibles que se desactivan para solo dejar desplazarse sobre el plano XZ.

```
if (keys[GLFW_KEY_W])
{
    if (terceraPersona == true) {
        if (position.y <= 5 and position.y >= 0) {
            position += front * velocity;
        }
        else {
            if (position.y >= 5) {
                position.y = 4.9f;
            }
            if (position.y <= -5) {
                position.y = 0.1f;
            }
        }
    }
    else {
        position += front * velocity;
    }
}

if (keys[GLFW_KEY_S])
{

    if (terceraPersona == true) {
        if (position.y <= 5 and position.y >= 0) {
            position -= front * velocity;
        }
        else {
            if (position.y >= 5) {
                position.y = 4.9f;
            }
            if (position.y <= 0) {
                position.y = 0.1f;
            }
        }
    }
    else {
        position -= front * velocity;
    }
}
if (keys[GLFW_KEY_A])
```

4. **Iluminación.**

Utilicé 2 skybox para representar la noche y el día.

```
//Skybox Dia
std::vector<std::string> skyboxFacesDia;
skyboxFacesDia.push_back("Textures/Skybox/mapa_2.tga");
skyboxFacesDia.push_back("Textures/Skybox/mapa_4.tga");
skyboxFacesDia.push_back("Textures/Skybox/mapa_dn.tga");
skyboxFacesDia.push_back("Textures/Skybox/mapa_up.tga");
skyboxFacesDia.push_back("Textures/Skybox/mapa_1.tga");
skyboxFacesDia.push_back("Textures/Skybox/mapa_3.tga");


//Skybox noche
std::vector<std::string> skyboxFacesNoche;
skyboxFacesNoche.push_back("Textures/Skybox/mapa-noche_2.tga");
skyboxFacesNoche.push_back("Textures/Skybox/mapa-noche_4.tga");
skyboxFacesNoche.push_back("Textures/Skybox/mapa-noche_dn.tga");
skyboxFacesNoche.push_back("Textures/Skybox/mapa-noche_up.tga");
skyboxFacesNoche.push_back("Textures/Skybox/mapa-noche_1.tga");
skyboxFacesNoche.push_back("Textures/Skybox/mapa-noche_3.tga");
```

Para el cambio automático de skybox me vi un poco limitado, ya que, a la hora de automatizar el cambio, después de 3 cambios en el skybox, la memoria comenzaba a desbordarse y el programa fallaba. Así que lo que hice fue implementar un método en el que se realiza el cambio de skybox tres veces por ejecución, El primer cambio a los 15 segundos de iniciar y el siguiente aproximadamente después de 40 segundos.

```
//cambiar el skybox despues de los primeros 15 segundos y después de 40segundos
if (cambioSkybox2 ==true && t0>=5000) {
    cambioSkybox = true;
}
if(cambioSkybox){
    if (t0 >= 750) {
        t0 = 0;
        if (skyboxTime == 0.0f) {//de mañana a noche
            skyboxTime = 2.0f;
            cambioSkybox = false;
            cambioSkybox2 = true;
            lucesNoche = true;
        }
        else if (skyboxTime == 2.0f) {//de noche a mañana
            skyboxTime = 0.0f;
            cambioSkybox=false;
            cambioSkybox2 = false;
            lucesNoche = false;
        }
    }
    if (skyboxTime == 0.0f) {
        skybox = Skybox(skyboxFacesDia);
    }
    if (skyboxTime == 2.0f) {
        skybox = Skybox(skyboxFacesNoche);
    }
}
```

La parte de las luces automáticas va de la mano con el cambio de skybox, ya que cuando se realiza el cambio, se modifican una serie de banderas que permiten realizar el apagado y cambio de luces.

```
if (lucesNoche) {
    spotLights[4].SetFlash(glm::vec3(40.0f, 15.0f, 10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[5].SetFlash(glm::vec3(40.0f, 15.0f, -10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[6].SetFlash(glm::vec3(30.0f, 15.0f, 10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[7].SetFlash(glm::vec3(30.0f, 15.0f, -10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[8].SetFlash(glm::vec3(20.0f, 15.0f, 10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[9].SetFlash(glm::vec3(20.0f, 15.0f, -10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
}
else {
    spotLights[4].SetFlash(glm::vec3(40.0f, -15.0f, 10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[5].SetFlash(glm::vec3(40.0f, -15.0f, -10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[6].SetFlash(glm::vec3(80.0f, 12.8f, 5.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[7].SetFlash(glm::vec3(100.0f, 12.8f, 0.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[8].SetFlash(glm::vec3(80.0f, 12.8f, -5.0f), glm::vec3(0.0f, -1.0f, 0.0f));
    spotLights[9].SetFlash(glm::vec3(20.0f, -15.0f, -10.0f), glm::vec3(0.0f, -1.0f, 0.0f));
}
```

De esta manera, cuando es de día, se prenden las luces en la casa de shrek y cuando es de noche, se iluminan los muñecos de nieve que se encuentran afuera de la casa.

Ejemplo de las luces dentro de la casa cuando es de día:

Para el apartado de luces que se puedan prender y apagar por medio de teclado, se utilizó el control de eventos con la tecla “Z” el cual se encuentra en el archivo Window.cpp.

```
//Para prender/apagar luces
if (key == GLFW_KEY_Z and action == GLFW_PRESS)
{
    if (theWindow->LucesOn == true) {
        theWindow->LucesOn = false;
    }
    else {
        theWindow->LucesOn = true;
    }
}
```

Luces encendidas:

Luces cuando es de noche:

5. Animación.

Animaciones básicas:

El par de águilas que se encuentran volando de un lado a otro es una animación básica ya que solo se mueven sobre el eje Z.

```
//****      Aves        *****
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(desplazamientoAve1));
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
if (avanzaA) {
    model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
else {
    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
ave1.RenderModel();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(desplazamientoAve2));
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
if (avanzaA) {
    model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
else {
    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
ave2.RenderModel();
```

```
//animación aves
if (posZave > -140 && avanzaA) {
    posZave -= 5.0 * deltaTime;
    //printf("%f\n", posXavion);
    avanza2 = true;
}
else {
    avanzaA = false;
    if (posZave < 0 && avanzaA == false) {
        posZave += 5.0 * deltaTime;
        //printf("%f\n", posXavion);
    }
    else { avanzaA = true; }
}
```

```
//Para animación de aves
desplazamientoAve1 = glm::vec3(ave1Inicio.x + posXave, ave1Inicio.y, ave1Inicio.z + posZave + mainWindow.getmuevexd());
desplazamientoAve2 = glm::vec3(ave2Inicio.x + posXave, ave2Inicio.y, ave2Inicio.z + posZave + mainWindow.getmuevexd());
// Clear the window
```

La puerta de la casa de shrek también es una animación básica, ya que solo se realiza el movimiento de rotación de 90 grados para abrir y cerrar la puerta. Esto se hace por medio del teclado. De manera interna se trabajó con jerarquía con la casa para poder hacer esta animación

```
//Casa
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(90.0f, -2.25f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
casaShrek.RenderModel();
//Puerta
model = modelaux;
model = glm::translate(model, glm::vec3(-23.0f, 4.0f, -2.9f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
model = glm::rotate(model, -10 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getpuerta()), glm::vec3(0.0f, 1.0f, 0.0f)); //Apertura de puerta
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
puertaCasaShrek.RenderModel();
```

La animación de la caminada de shrek también la considero una animación básica, ya que solo se desplaza sobre el eje x, aunque se realizan rotaciones para mover pies y brazos, pero no siguen una ecuación matemática o algoritmo en concreto.

Esta animación se lleva acabo presionando la telca “Y”.

Animaciones complejas.

La pequeña carrera que realizan las dos carretas, una jalada por burro y otra por un puerco, realizan dos curvas, siguiendo una trayectoria dada por una función de Bezier, que calcula una curva dándole un punto inicial, uno final y uno en medio, posteriormente, avanzan en línea recta y se detienen simulando un derrape.

Carreta de burro:

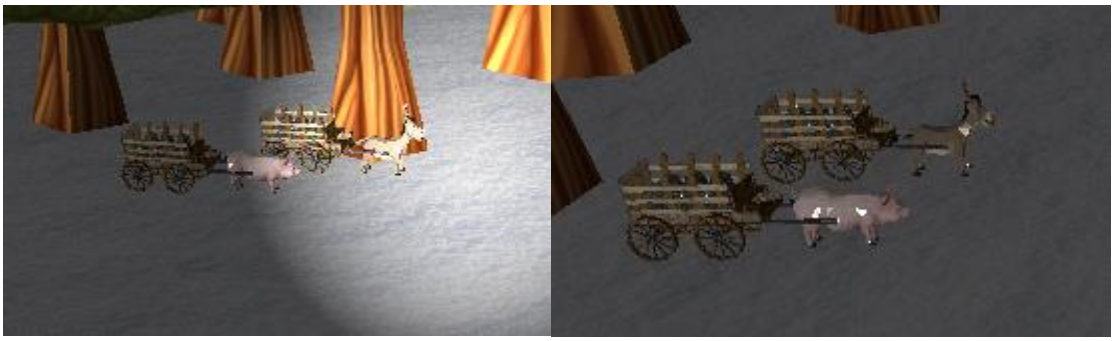
```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(desplazamientoCarreta1));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));

if (despuesVuelta == false and terminaCarreta1 == false) {
    if (activoCarreta1 == 0.0f) {
        model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
        posXCarreta1 = posXCarreta1 - 0.2;
    }
    desplazamientoCarreta1 = glm::vec3(carreta1Inicio.x + xBezierCarreta1, carreta1Inicio.y, carreta1Inicio.z + zBezierCarreta1 + posXCarreta1);
}
else if (activoCarreta1 == 0.0f and terminaCarreta1 == false) {
    if (desplazamientoCarreta1.x < 100) {
        model = glm::rotate(model, topeRotacionCarreta1 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    }
    if (despuesVueltaCarreta2 == true) {
        posYCarreta1 = posYCarreta1 + 1.2;
        desplazamientoCarreta1 = glm::vec3(carreta1Inicio.x + xBezierCarreta1 + posYCarreta1, carreta1Inicio.y, carreta1Inicio.z + zBezierCarreta1 + posXCarreta1);
    }
}
if (desplazamientoCarreta1.z <= 80.0f and cuentaCarreta1 < 1 and terminaCarreta1 == false) {
    //Funciones de bezier para trazar las curvas
    activoCarreta1 = 1.0f;
    cuentaCarreta1 = cuentaCarreta1 + 0.01;
    xaCarreta1 = getPt(0.0f, 7.0f, cuentaCarreta1);
    yaCarreta1 = getPt(0.0f, -30.0f, cuentaCarreta1);
    xbCarreta1 = getPt(7.0f, 20.0f, cuentaCarreta1);
    ybCarreta1 = getPt(-30.0f, -35.0f, cuentaCarreta1);

    // The Black Dot
    xBezierCarreta1 = getPt(xaCarreta1, xbCarreta1, cuentaCarreta1);
    zBezierCarreta1 = getPt(yaCarreta1, ybCarreta1, cuentaCarreta1);

    model = glm::rotate(model, topeRotacionCarreta1 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    topeRotacionCarreta1 -= 0.8;

    if (topeRotacionCarreta1 <= 90) {
        topeRotacionCarreta1 = 90;
    }
}
if (cuentaCarreta1 > 0.99 and activoCarreta1 == 1.0) {
    activoCarreta1 = 0.0f;
    adelanteCarreta1 = false;
    despuesVuelta = true;
    topeRotacionCarreta1 = 90.0;
    //Sonido de las carretas.
    irrklang::ISoundEngine* sonidoCarreta = createIrrKlangDevice();
    sonidoCarreta->play2D("audio/carreta.mp3");
}
```

```
if (desplazamientoCarreta1.x >= 0) {
    if (terminaCarreta1 == false) {
        topeRotacionCarreta1 = topeRotacionCarreta1 - 1.0;
        model = glm::rotate(model, topeRotacionCarreta1 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    }
    if (topeRotacionCarreta1 == 45.0) {
        terminaCarreta1 = true;
        model = glm::rotate(model, 45.0f * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    }
}


glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
carreta.RenderModel();
```

Carreta del puerco:

```
//carreta puerco
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(desplazamientoCarreta2));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
if (despuesVueltaCarreta2 == false and terminaCarreta2 == false) {
    if (activoCarreta2 == 0.0f) {
        model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
        posXCarreta2 = posXCarreta2 - 0.2;
    }
    desplazamientoCarreta2 = glm::vec3(carreta2Inicio.x + xBezierCarreta2, carreta2Inicio.y, carreta2Inicio.z + zBezierCarreta2 + posXCarreta2);

}
else if (activoCarreta2 == 0.0f and terminaCarreta2 == false) {
    if (desplazamientoCarreta2.x < 100) {
        model = glm::rotate(model, topeRotacionCarreta2 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    }
    posYCarreta2 = posYCarreta2 + 1.2;
    desplazamientoCarreta2 = glm::vec3(carreta2Inicio.x + xBezierCarreta2 + posYCarreta2, carreta2Inicio.y, carreta2Inicio.z + zBezierCarreta2 + posXCarreta2);
}
if (desplazamientoCarreta2.z <= 70.0f and cuentaCarreta2 < 1 and terminaCarreta2 == false) {
    //Funciones de bezier para trazar las curvas
    activoCarreta2 = 1.0f;
    cuentaCarreta2 = cuentaCarreta2 + 0.01;
    xaCarreta2 = getPt(0.0f, 7.0f, cuentaCarreta2);
    yaCarreta2 = getPt(0.0f, -30.0f, cuentaCarreta2);
    xbCarreta2 = getPt(7.0f, 20.0f, cuentaCarreta2);
    ybCarreta2 = getPt(-30.0f, -35.0f, cuentaCarreta2);

    // The Black Dot
    xBezierCarreta2 = getPt(xaCarreta2, xbCarreta2, cuentaCarreta2);
    zBezierCarreta2 = getPt(yaCarreta2, ybCarreta2, cuentaCarreta2);

    model = glm::rotate(model, topeRotacionCarreta2 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    topeRotacionCarreta2 -= 0.8;

    if (topeRotacionCarreta2 <= 90) {
        topeRotacionCarreta2 = 90;
    }
}
if (cuentaCarreta2 > 0.99 and activoCarreta2 == 1.0) {
    activoCarreta2 = 0.0f;
    adelanteCarreta2 = false;
    despuesVueltaCarreta2 = true;
    topeRotacionCarreta2 = 90.0;

}
```

```
if (desplazamientoCarreta2.x >= 0) {
    if (terminaCarreta2 == false) {
        topeRotacionCarreta2 = topeRotacionCarreta2 - 1.0;
        model = glm::rotate(model, topeRotacionCarreta2 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    }
    if (topeRotacionCarreta2 == 45.0) {
        terminaCarreta2 = true;
        model = glm::rotate(model, 45.0f * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    }
}


glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
carreta2.RenderModel();
```

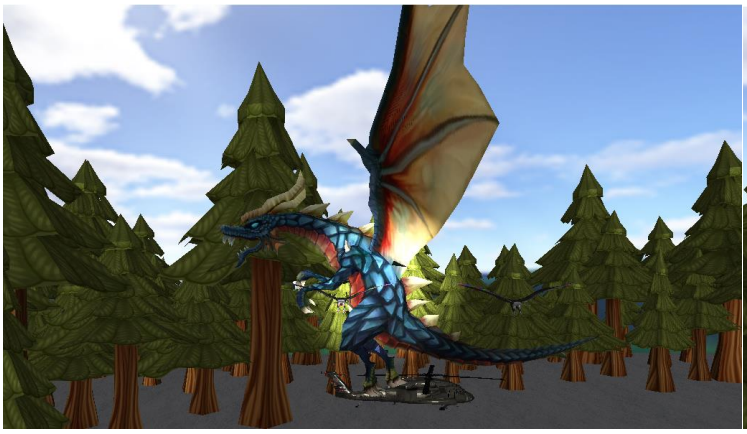


Otra animación compleja es la de el dragón, ya que este se desplaza a lo largo del eje x pero con la peculiaridad de que en el eje Y, sigue una trayectoria senoidal para simular el vuelo de un dragon. Además, se implementó la animación de las alas, lo que implicó trabajar con jerarquía.

```
//animación dragon
if (posXdragon > -140 && avanzad) {
    posXdragon -= 10.0 * deltaTime;
    //printf("%f\n", posXavion);
    avanza = true;
}
else {
    avanzad = false;
    if (posXdragon < 0 && avanzad == false) {
        posXdragon += 10.0 * deltaTime;
        //printf("%f\n", posXavion);
    }
    else { avanzad = true; }
}
```

```
//Para animación de Dragon
offset += 1.0;
posYdragon = 2 * sin(2 * offset * toRadians);
desplazamientoDragon = glm::vec3(dragonInicio.x + posXdragon + mainWindow.getmuevexd(), dragonInicio.y + posYdragon + mainWindow.getmueveyd(), dragonInicio.z + mainWindow.getmuevezd());
```

```
//************      Dragon      ***********
//Animación de alas

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(desplazamientoDragon));
modelauxD = model;
model = glm::scale(model, glm::vec3(0.40f, 0.40f, 0.40f));
if (avanzad) {
    model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
else {
    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
dragon.RenderModel();
//Alas del dragon
    //Ala izquierda
model = modelauxD;
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.40f, 0.40f, 0.40f));
if (avanzad) {
    model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::rotate(model, rotacionDragon * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
else {
    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::rotate(model, rotacionDragon * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
dragonAL.RenderModel();
    //Ala derecha
model = modelauxD;
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.40f, 0.40f, 0.40f));
if (avanzad) {
    model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::rotate(model, -1*rotacionDragon* toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
else {
    model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::rotate(model, -1*rotacionDragon * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
}
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
dragonAR.RenderModel();
```



Animación por keyframes.

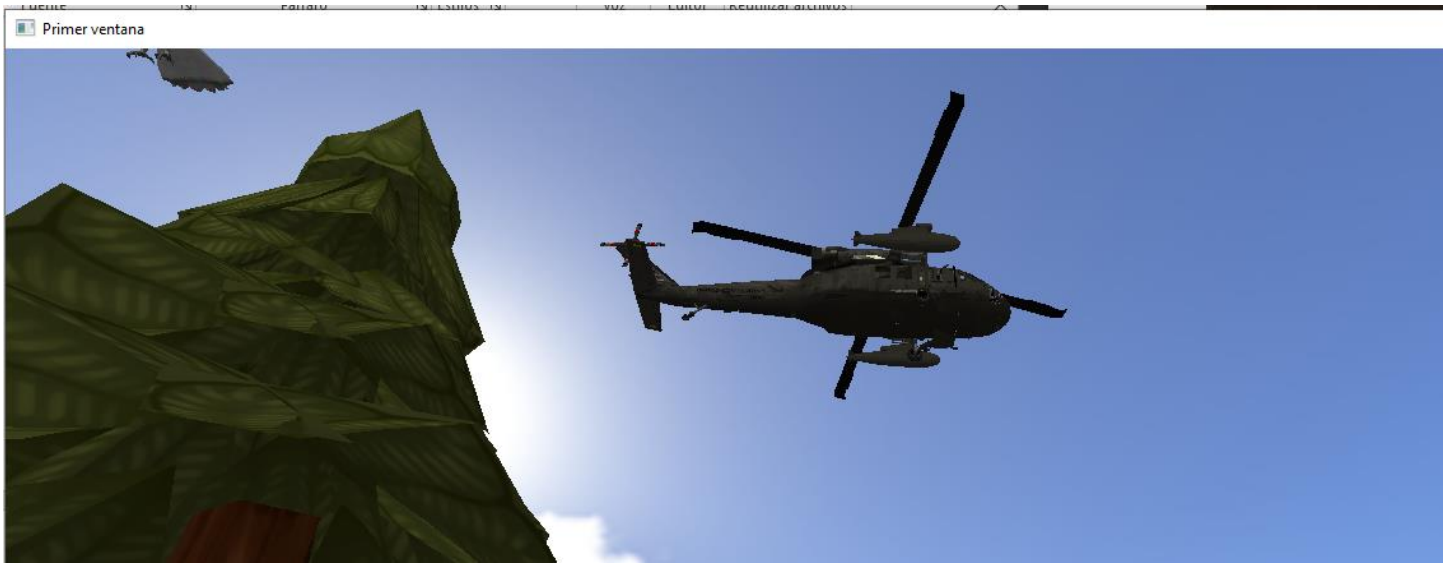
En este apartado, aprovechando que es requisito que el modelo del helicóptero con la jerarquía de la hélice se incluya en el proyecto, decidí animar el helicóptero para realizar un recorrido largo en casi todo el mapa. Este recorrido es algo más complejo que lo que se vio en laboratorio, ya que implementé varios giros de la aeronave.

En total fueron necesarios 103 frames para realizar la animación por medio de keyframes.

```
76      //KEYFRAMES DECLARADOS INICIALES
77      KeyFrame[0].movAvion_x = 0.0f;
78      KeyFrame[0].movAvion_y = 0.0f;
79      KeyFrame[0].giroAvion = 0;
80      KeyFrame[0].giroHelice = 0;
81
82      KeyFrame[1].movAvion_x = 0.0f;
83      KeyFrame[1].movAvion_y = 0.0f;
84      KeyFrame[1].giroAvion = 0;
85      KeyFrame[1].giroHelice = 360;
86
87      KeyFrame[2].movAvion_x = 0.0f;
88      KeyFrame[2].movAvion_y = 0.0f;
89      KeyFrame[2].giroAvion = 0;
90      KeyFrame[2].giroHelice = 720;
91
92      KeyFrame[3].movAvion_x = 0.0f;
93      KeyFrame[3].movAvion_y = 0.0f;
94      KeyFrame[3].giroAvion = 0;
95      KeyFrame[3].giroHelice = 1080;
96
97      KeyFrame[4].movAvion_x = 0.0f;
98      KeyFrame[4].movAvion_y = 0.0f;
99      KeyFrame[4].giroAvion = 0;
00      KeyFrame[4].giroHelice = 1440;
01
```

```
1338    //Aterriza
1339    KeyFrame[97].movAvion_x = 0.0f;
1340    KeyFrame[97].movAvion_y = 75.0f;
1341    KeyFrame[97].movAvion_z = 0.0f;
1342    KeyFrame[97].giroAvion = 360;
1343    KeyFrame[97].giroHelice = 1800;
1344
1345    KeyFrame[98].movAvion_x = 0.0f;
1346    KeyFrame[98].movAvion_y = 60.0f;
1347    KeyFrame[98].movAvion_z = 0.0f;
1348    KeyFrame[98].giroAvion = 360;
1349    KeyFrame[98].giroHelice = 2160;
1350
1351    KeyFrame[99].movAvion_x = 0.0f;
1352    KeyFrame[99].movAvion_y = 45.0f;
1353    KeyFrame[99].movAvion_z = 0.0f;
1354    KeyFrame[99].giroAvion = 360;
1355    KeyFrame[99].giroHelice = 2520;
1356
1357    KeyFrame[100].movAvion_x = 0.0f;
1358    KeyFrame[100].movAvion_y = 30.0f;
1359    KeyFrame[100].movAvion_z = 0.0f;
1360    KeyFrame[100].giroAvion = 360;
1361    KeyFrame[100].giroHelice = 2880;
1362
1363    KeyFrame[101].movAvion_x = 0.0f;
1364    KeyFrame[101].movAvion_y = 15.0f;
1365    KeyFrame[101].movAvion_z = 0.0f;
1366    KeyFrame[101].giroAvion = 360;
1367    KeyFrame[101].giroHelice = 3240;
1368
1369    KeyFrame[102].movAvion_x = 0.0f;
1370    KeyFrame[102].movAvion_y = 0.0f;
1371    KeyFrame[102].movAvion_z = 0.0f;
1372    KeyFrame[102].giroAvion = 360;
1373    KeyFrame[102].giroHelice = 3600;
1374
```

```
418   #define MAX_FRAMES 110 //variarlo a 100 o 200 dependiendo de la complejidad de las animaciones
419   int i_max_steps = 90; //cuantos datos voy a interpolar entre cuadro y cuadro
420   int i_curr_steps = 103; //
421   typedef struct _frame
422   { //por cada variable que se declare, se debe una segunda variable para guardar la información de las interpolaciones. para usar el valor actual y el valor anterior
423       //Variables para GUARDAR Key Frames
424       float movAvion_x;       //Variable para PosicionX
425       float movAvion_y;       //Variable para PosicionY
426       float movAvion_z;
427       float movAvion_xInc;        //Variable para IncrementoX
428       float movAvion_yInc;        //Variable para IncrementoY
429       float movAvion_zInc;        //Variable para IncrementoY
430       float giroAvion;
431       float giroAvionInc;
432       float giroHelice;
433       float giroHeliceInc;
434   }FRAME;
435
436   FRAME KeyFrame[MAX_FRAMES];
437   int FrameIndex = 103;           //introducir datos. debe ser el mismo valor que i_curr_steps
438   bool play = false; //para poer reproducir por primera vez
439   int playIndex = 0;
440
```


Primer ventana


Primer ventana

6. Audio

Realizando la correspondiente investigación, me encontré con una biblioteca muy útil para implementar efectos y pistas de sonido en proyectos de OpenGL.

El lugar de donde la obtuve es el siguiente: https://www.ambiera.com/irrklang/

Leyendo un poco la documentación descubrí como importar pistas de audio para poder implementar el sonido de fondo y algunos sonidos en puntos específicos para poder simular el efecto.

```
21
22      #include <irrklang/irrKlang.h>
23      using namespace irrklang;
24      #pragma comment(lib, "irrKlang.lib")
25
```

La primera aplicación que le di es la de colocarle al proyecto el sonido de fondo, lo cual decidí colocar una canción navideña ya que la temática se trata de eso.

```
//Para el soundtrack del juego. El true es para que haga ciclo infinito y reinicie una vez que termina.
irrklang::ISoundEngine* sonidoFondo = createIrrKlangDevice();
sonidoFondo->play2D("audio/navidad.mp3", true);
```

Para trabajar de manera más cómo, implementé un sistema para reproducir el sonido de fondo por medio del teclado. Utilizando la tecla “V” se empezará a reproducir el sonido de fondo, pero sin la opción de reproducción automática infinita. Solo se reproduce una vez. Para volver a reproducirlo es necesario presionar una vez la tecla “V” para reiniciar banderas y nuevamente la tecla “V” para reproducir el sonido.

La segunda aplicación que le di fue la de darle efecto de sonido a las carretas del burro y el puerco, en el momento en el que corren:

```
2714
2715            if (topeRotacionCarreta1 <= 90) {
2716                topeRotacionCarreta1 = 90;
2717            }
2718        }
2719        if (cuentaCarreta1 > 0.99 and activoCarreta1 == 1.0) {
2720            activoCarreta1 = 0.0f;
2721            adelanteCarreta1 = false;
2722            despuesVuelta = true;
2723            topeRotacionCarreta1 = 90.0;
2724            //Sonido de las carretas.
2725            irrklang::ISoundEngine* sonidoCarreta = createIrrKlangDevice();
2726            sonidoCarreta->play2D("audio/carreta.mp3");
2727        }
```

La tercera aplicación que le di fue para dar el efecto a la animación del helicóptero, pues una vez se comienza a ejecutar la animación, se reproduce el sonido.

```
void inputKeyframes(bool* keys)
{ //cuando se presiona la barra espaciadora se ejeuta esta parte
    if (keys[GLFW_KEY_SPACE])
    {
        //Sonido de helicoptero.
        irrklang::ISoundEngine* sonidoHelicoptero = createIrrKlangDevice();
        sonidoHelicoptero->play2D("audio/Helicoptero.mp3");
        if(reproduciranimacion<1)
```

La cuarta implementación que le di fue como efecto de sonido de las águilas que están planeando en el mapa. Cada que comienzan el recorrido, se reproduce el sonido de manera automática.

```
//animación aves
if (posZave > -140 && avanzaA) {
    posZave -= 5.0 * deltaTime;
    avanza2 = true;
    //efecto de sonido
    if (sonidoAv == true) {
        irrklang::ISoundEngine* sonidoAves = createIrrKlangDevice();
        sonidoAves->play2D("audio/sonidoAguilas.mp3");
        sonidoAv = false;
    }
}
else {
    avanzaA = false;
    sonidoAv = true;
    if (posZave < 0 && avanzaA == false) {
        posZave += 5.0 * deltaTime;
    }
    else { avanzaA = true; }
}
//Para animación de Dragon
```

BIBLIOGRAFÍA Y RECURSOS.

- **Biblioteca de audio - https://www.ambiera.com/irrklang/**
- **Repositorio de modelos 3d gratuitos con licencia para uso académico -** http://rigmodels.com/index.php
- **Blender.**
- **GIM.**
- **Microsoft Visual Studio.**